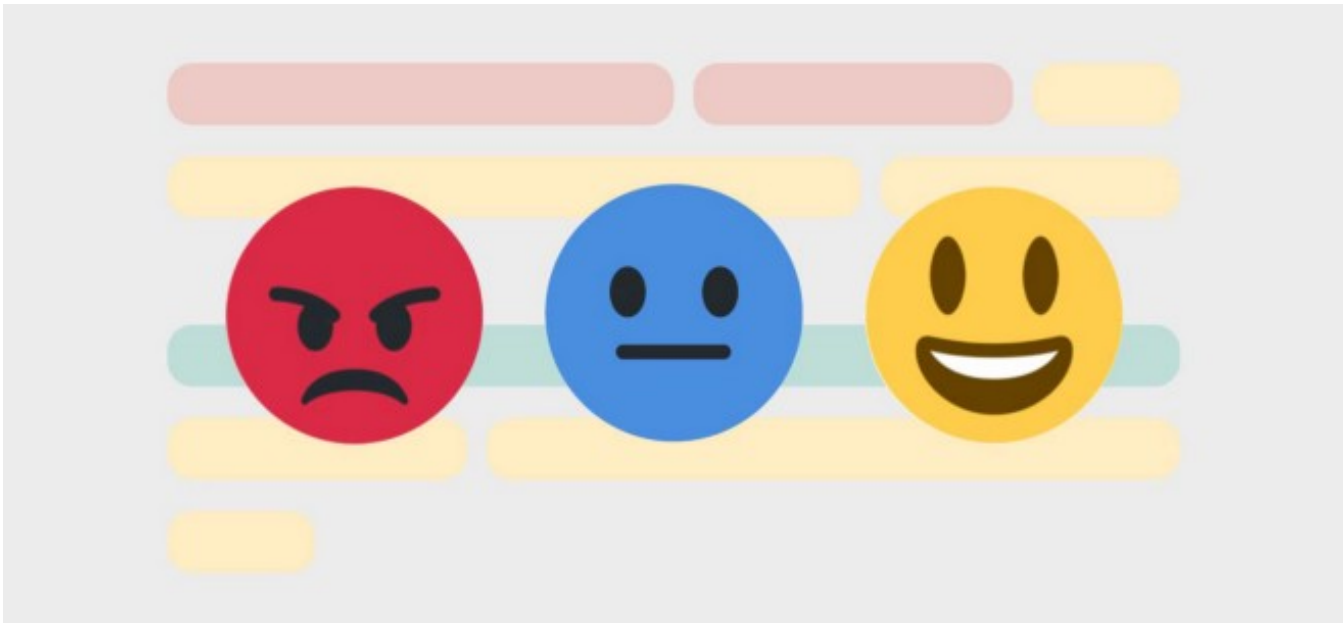
Marlon Sousa

4 Followers About

Análise de Sentimentos — Natural Language Processing

Marlon Sousa 3 days ago · 4 min read

Análise de Sentimentos usando um algoritmo de CNN e Rede Neural



Preprocessamento

Importações

```
# __Author__ = Marlon Sousa
# __Blog__ = marlonsousa.medium.com

import numpy as np
import math
import re
import pandas as pd
from bs4 import BeautifulSoup
```

```
import spacy as sp
import string
import random
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras import layers
```

Definimos os nomes das colunas e lemos nosso dataset com pandas

```
cols = ["sentiment", "id", "date", "query", "user", "text"]

train_data = pd.read_csv("trainingandtestdata/train.csv", header=None, engine="python", encoding="latin1", names=cols)

test_data = pd.read_csv("trainingandtestdata/test.csv", header=None, engine="python", encoding="latin1", names=cols)
data = train_data
data.drop(["id", "date", "query", "user"], axis=1, inplace=True)
data.head()
X = data.iloc[:, 1].values
y = data.iloc[:, 0].values
```

Fazemos o split dos dados entre treino e teste

```
from sklearn.model_selection import train_test_split

X, _, y, _ = train_test_split(X, y, test_size=0.85, stratify=y)
unique, counts = np.unique(y, return_counts=True)
```

Criamos uma função para limpar nossas frases, estas frases serão usadas no nosso algoritmo, para isso não podemos deixar simbolos ou textos que não fazem sentido. Com isso precisamos fazer o preprocessamento dos nosso dados.

```
def clean_tweets(tweet):
    tweet = BeautifulSoup(tweet, "lxml").get_text()
    tweet = re.sub(r"@[A-Za-z0-9]+", " ", tweet)
    tweet = re.sub(r"https?://[A-Za-z0-9./]+", " ", tweet)
    tweet = re.sub(r"[^a-zA-Z.!?]", " ", tweet)
    tweet = re.sub(r" +", " ", tweet)
    return tweet
```

Utilizando a lib do Spacy “puxamos” as “Stop Words” que fará mais um preprocessamento nas palavras pouco usadas.

Criamos mais uma função para o preprocessamento, e iremos fazer replace de algumas palavras.

```
def clean_tweets2(tweet):
    tweet = tweet.lower()
    document = nlp(tweet)

    words = []
    for token in document:
        words.append(token.text)

    words = [word for word in words if word not in stop_words and word not in string.punctuation]
    words = ' '.join([str(element) for element in words])

    return words


data_clean = [clean_tweets2(clean_tweets(tweet)) for tweet in X]

for _ in range(10):
    print(data_clean[random.randint(0, len(data_clean)) - 1])
```

```
data_labels = y

data_labels[data_labels == 4] = 1


#np.unique(data_labels)
```

Tokenização

Fazemos a tokenização utilizando o Tensorflow. Para isto criamo um “vocabulário” que a máquina e o algoritmo consiga entender.

As palavras serão substituidas por número para que o algoritmo funcione e faça os cálculos. Logo após utilizamos o Pad Sequences do Preprocessamento para preencher valores de frases que ficaram curtas.

```
import tensorflow_datasets as tfds
```

```
text = tokenizer.decode(ids)

data_inputs = [tokenizer.encode(setence) for setence in data_clean]

for _ in range(10):
    print(data_inputs[random.randint(0, len(data_inputs)) - 1])



max_len = max([len(setence) for setence in data_inputs])

data_inputs = tf.keras.preprocessing.sequence.pad_sequences(data_inputs,
                                                            value=0,
                                                            padding='post',
                                                            maxlen=max_len)

for _ in range(10):
    print(data_inputs[random.randint(0, len(data_inputs)) - 1])
```

```
ids = tokenizer.encode("i am happy")
ids
```

```
[269, 3787, 371]
```

A frase “i am happy” teve cada palavra convertida para um número

Construção do Modelo

Aqui faremos a construção do Modelo.

Começamos criando três(3) camadas de Convolução de uma(1) dimensão, passamos o parâmetro de filtros que serão utilizados, o tamanho do kernel a ser utilizando, o padding e nossa ativação será a função relu.

Logo Após faremos o Max Pooling em uma(1) dimensão.
Após criaremos finalmente nossa Rede Neural Densa com a ativação da função relu novamente e por último faremos uma camada de Dropout para que evite Overfitting na nossa rede neural.

```
# Construção do Modelo Base
class DCNN(tf.keras.Model):

    def __init__(self, vocab_size, emb_dim=128, nb_filters=50, ffn_units=512, nb_classes=2, dropout_rate=0.1, training=False, name="dcnn"):
        super(DCNN, self).__init__(name=name)

        #Camada de Convolução
```

```
        self.pool = layers.GlobalMaxPooling1D()

        #Camada Densa
        self.dense_1 = layers.Dense(units=ffn_units, activation='relu')
        self.dropout = layers.Dropout(rate=dropout_rate)

        if nb_classes == 2:
            self.last_dense = layers.Dense(units=1, activation='sigmoid')
        else:
            self.last_dense = layers.Dense(units=nb_classes, activation='softmax')
```

Fazemos uma função ainda na classe para chamar a configuração já setada

```
def call(self, inputs, training):
    x = self.embedding(inputs)
    x_1 = self.bigram(x)
    x_1 = self.pool(x_1)
    x_2 = self.trigram(x)
    x_2 = self.pool(x_2)
    x_3 = self.fourgram(x)
    x_3 = self.pool(x_3)

    merged = tf.concat([x_1, x_2, x_3], axis=-1) #Batch_size, 3*nb_filters
    merged = self.dense_1(merged)
    merged = self.dropout(merged, training)
    output = self.last_dense(merged)

    return output
```

Definimos os parâmetros para usarmos a nossa rede neural

```
vocab_size = tokenizer.vocab_size
emb_dim = 200
nb_filters = 100
ffn_units = 256
nb_classes = len(set(train_labels))
batch_size = 64
dropout_rate = 0.2
nb_epochs = 10

Dcnn = DCNN(vocab_size=vocab_size, emb_dim=emb_dim, nb_filters=nb_filters, ffn_units=ffn_units,
            nb_classes=nb_classes, dropout_rate=dropout_rate)
```

Após tudo isso faremos o treino das variáveis e salvaremos os pesos para que não precisemos fazer o treino novamente

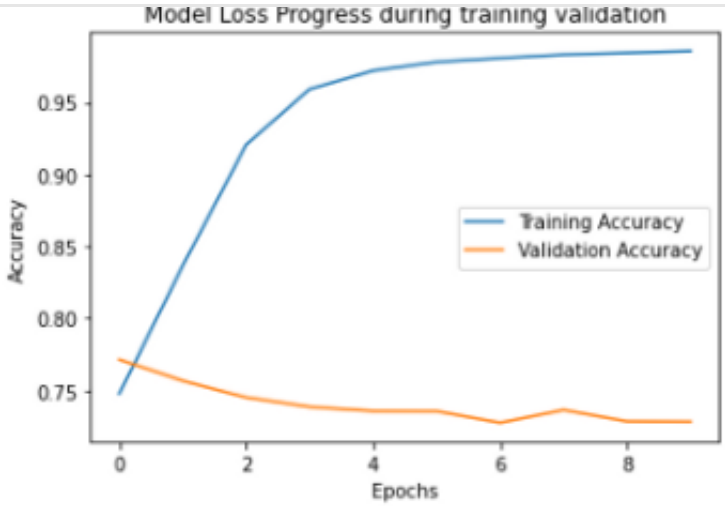
```
Dcnn.save_weights("Weights.h5")


Dcnn.summary()
```

Faremos uma matriz de confusão para vermos a função de loss e accuracy

```
results = Dcnn.evaluate(test_inputs, test_labels, batch_size=batch_size)
print(results)

y_pred_test = Dcnn.predict(test_inputs)


y_pred_test = (y_pred_test > 0.5)

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(test_labels, y_pred_test)
cm

sns.heatmap(cm, annot=True);

history.history.keys()
```

```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.title("Model Loss Progress during training validation")
plt.xlabel("Epochs")
plt.ylabel("Losses")
plt.legend(["Training Loss", "Validation Loss"]);
```


```
plt.plot(history.history["accuracy"])
plt.plot(history.history["val_accuracy"])
plt.title("Model Loss Progress during training validation")
```



E podemos fazer nosso próprios testes

```
text = "I love you so much"
text = tokenizer.encode(text)


print(Dcnn(np.array([text]), training=False).numpy())
```

```
text = "I hate you"
text = tokenizer.encode(text)
```

```
text
```

```
[52510, 52469, 64, 3293]
```

```
Dcnn(np.array([text]), training=False).numpy()
```

```
array([[1.7763492e-11]], dtype=float32)
```

```
text = "I love you so much"
text = tokenizer.encode(text)
text
```

```
[52510, 52469, 13, 4280, 3918, 14667]
```

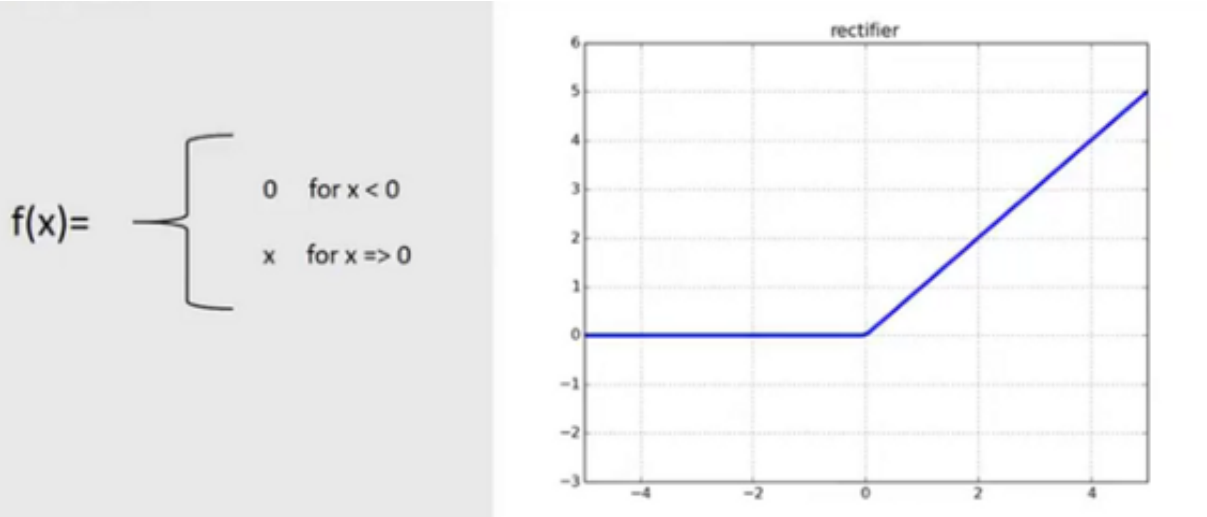
```
Dcnn(np.array([text]), training=False).numpy()
```

```
array([[0.99999404]], dtype=float32)
```

```
print(Dcnn(np.array([text]), training=False).numpy())
```

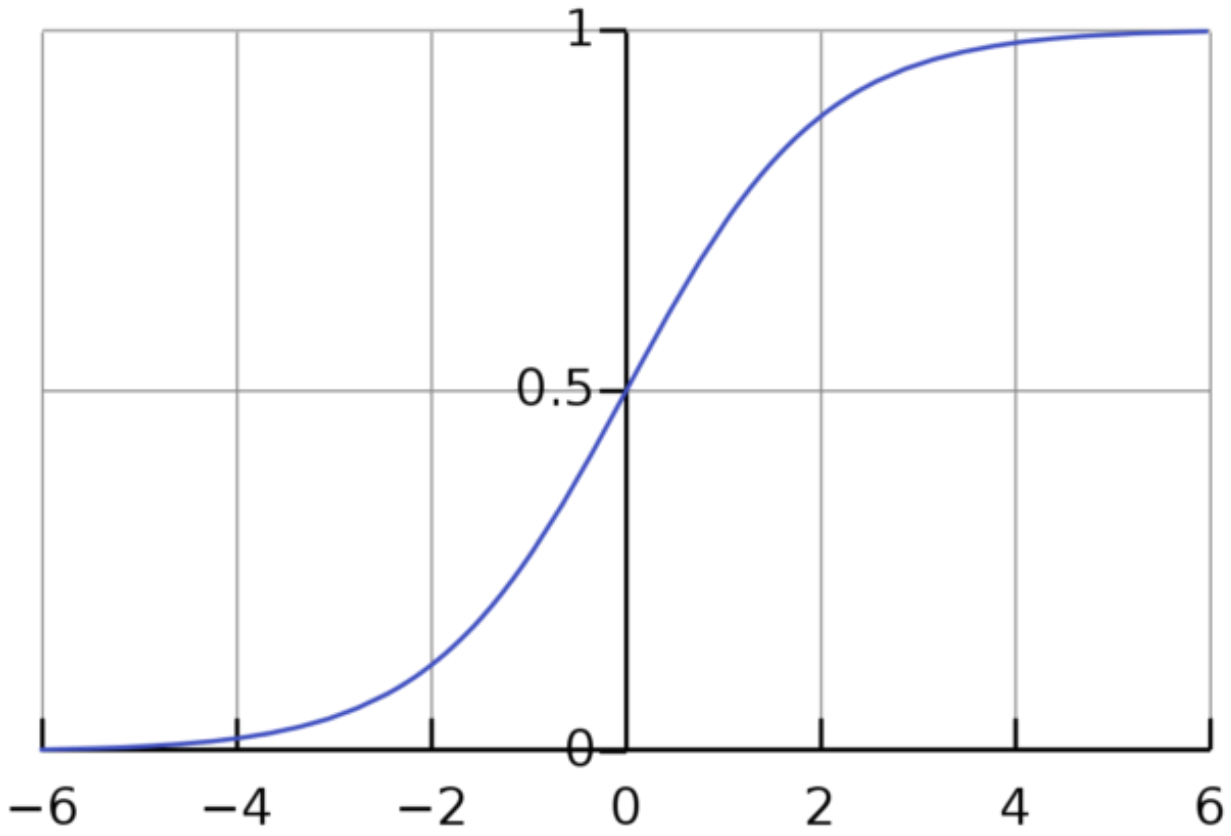
```
[[0.99999404]]
```

Funções de Ativação

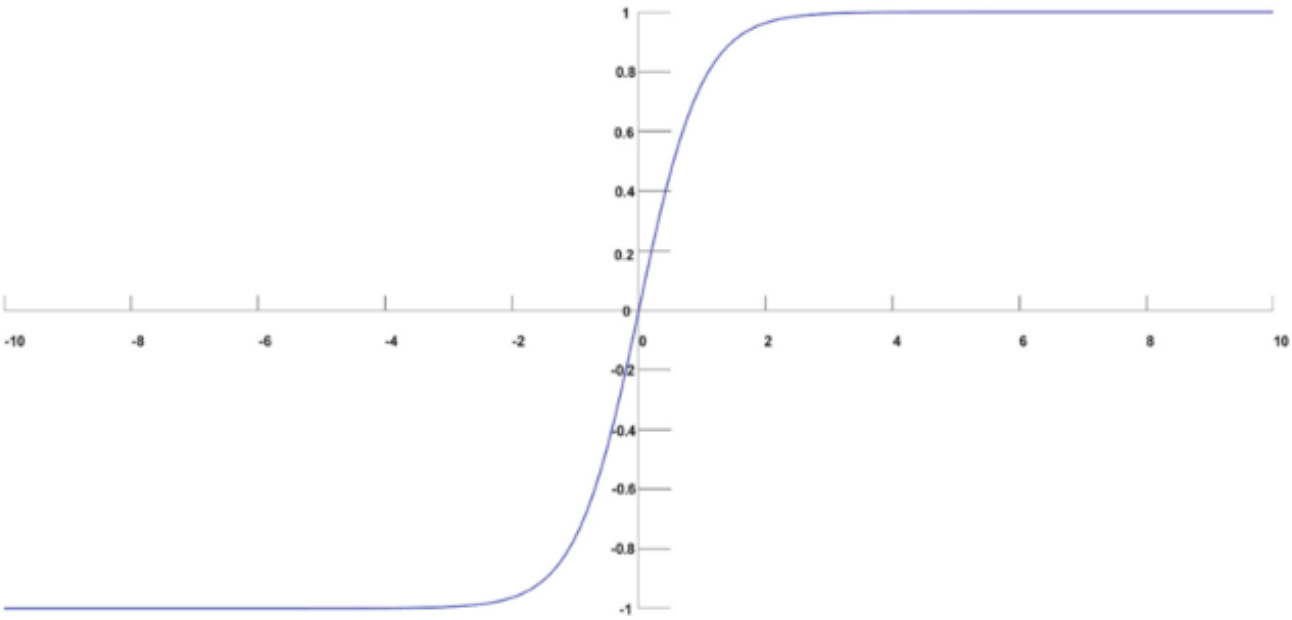


Função RELU nas camadas de convolução

$$S(x) = \frac{1}{1 + e^{-x}}$$



$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$



Função Softmax para mais de 2 classes

Naturallanguageprocessing Artificial Intelligence Machine Learning Pyhton